Quantum Machine Learning with Tensor Networks

Arjun Raje

September 2022

1 Introduction

Quantum computing is a rapidly growing field that has attracted considerable interest due to the possibility of performing hard classical tasks more efficiently using quantum devices. The discovery of quantum algorithms like Grover's and Shor's algorithms which outperform their classical counterparts demonstrated that quantum computers could potentially accomplish tasks beyond the reach of classical computers.

Currently, quantum devices are in the Noisy Intermediate Scale Quantum (NISQ) era, characterized by faulty and low fidelity devices. Grover's and Shor's algorithms can't be implemented on quantum devices because of the poor device performance, which opens an important question in the field — can one develop quantum algorithms that work on NISQ devices?

One exciting development in the field is quantum machine learning, where one solves classical machine learning tasks using quantum methods. The success of modern machine learning across a variety of fields, such as natural language and image processing, suggests that quantum advantages would lead to significant impacts.

Motivated by this prospect, researchers have made advances in quantum machine learning across a variety of machine learning tasks, from the introduction of quantum convolutional neural networks to variational quantum classifiers, and kernel-based quantum methods.

However, many quantum machine learning algorithms suffer from the aforementioned problem — they cannot be tested on NISQ hardware. A useful strategy is to use tensor networks (factorization of high-dimensional tensors, see Section 2), which can be stored on a regular computer and easily translated to a quantum circuit. Thus, one can perform benchmark machine learning tasks using tensor networks on a regular computer, and be confident they will translate to a quantum device (potentially at larger scale).

Researchers discovered that tensor networks are able to classify simple image datasets at the same accuracy as state-of-the-art algorithms. However, tensor networks have not been applied to more complicated datasets, so in this paper, we apply these methods to a more complex dataset, CIFAR-10.

2 Background

2.1 Quantum computers

Classical computers function by composing algorithms from bits, which take the value of either 0 or 1. Instead of using bits, a quantum computer uses qubits, so when it takes in data we have to represent everything in terms of qubits. Qubits are not like traditional bits in the way that bits only take the value of 0 and 1, meaning there are only two possible states. Instead of 0 or 1, a qubit can store a linear combination state (e.g., $a\,|0\rangle + b\,|1\rangle$), and the coefficients correspond to probabilities where $|a|^2$ is the probability of a $|0\rangle$ state being measured and $|b|^2$ is the probability of a $|1\rangle$ state being measured.

In a quantum computer, we encode data using experimental platforms where quantum mechanics controls the dynamics — for example, very cold atoms. Any "quantum system" can in principle work, and researchers have used superconducting circuits, neutral atoms, ions, etc. In every case, there are two possible quantum states, which can be regarded as two possible configurations of the system. In quantum information, we generally abstract from the experimental quantum computer and instead use the rules of quantum mechanics — which are the same for every platform — to define algorithms and experiments.

Because quantum computers operate under a different paradigm than classical computers, it is reasonable to think that quantum computers may be able to solve tasks that classical computers can't. In particular, quantum computers seem more expressive than classical computers. If you have two bits, you can only store four states: $(0,0), (0,1), (1,0)$, and $(1,1)$. On the other hand, two qubits can in principle store many more states because the linear combinations are continuous, not discrete. However, to distinguish similar linear combinations we have to measure them many times to get an accurate measurement of the quantum states. The field of quantum information is thus defined by trying to use the rules of quantum mechanics to create novel algorithms that have advantages over classical methods.

2.2 Tensor networks

Tensor networks are used to decompose a large tensor with a high order, into tensors with smaller dimensions that can be contracted together to get the original large tensor. To represent a tensor network, we can use tensor diagram notation.

The tensor network we use as our data structure is a Matrix Product State (MPS) because it can be directly mapped to a quantum circuit on a quantum computer with very little effort, and an MPS has been proven to show great results for classifying data. An MPS is an estimation of a tensor with N indices by breaking it down into N tensors, each with only 3 indices, making computation much easier. Using tensor diagram notation, an MPS can be represented like in Fig. 3.

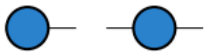

Figure 1: Vector and matrix representation

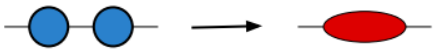

Figure 2: Matrix Multiplication

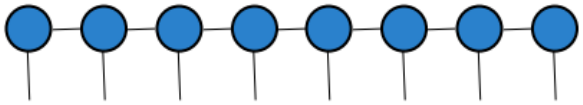

Figure 3: MPS representation in tensor diagram notation

Figure 4: Tensor Diagram Notation

3 Methods

We want to find out if quantum machine learning can classify complex images, so we tested our algorithm on MNIST and FashionMNIST as a benchmark and finally on CIFAR-10.

MNIST is a basic black and white dataset that contains handwritten images that need to be classified as any number from 0 to 9. FashionMNIST is also black and white, but the images are 10 different pieces of clothing. Both MNIST and FashionMNIST consist of 28x28 grayscale images, totaling 784 pixels. CIFAR-10, on the other hand, contains 32x32 color images (1024 pixels per image), in one of 10 categories that include certain animals and vehicles.

The classifier used is a Matrix Product State with 2-dimensional visible indices, and the bond dimension can be adjusted to maximize accuracy and speed. The MPS also includes a label index whose dimension is the number of classes. Once we contract the MPS, we obtain a vector, and we take the logarithmic softmax of the vector to determine which class the image was classified into.

We initialize the MPS such that each tensor is an identity matrix perturbed with small noise (1×10^{-4}). This avoids the problem of vanishing gradients or degenerate predictions.

Images are encoded into quantum states using the following transformation:

$$|\psi\rangle = \cos\left(\frac{\pi}{2}x\right)|0\rangle + \sin\left(\frac{\pi}{2}x\right)|1\rangle \tag{1}$$

This outputs a 2D vector for each pixel, transforming the image into a 2×784 or 2×1024 matrix, depending on the dataset.

Once encoded, we apply the MPS forward function, which contracts the tensors along bond dimensions to produce a vector of size 10. We apply a log-softmax function to convert this into class probabilities, calculate the loss

via cross-entropy, and update the MPS parameters using the Adam optimizer with a learning rate of 1×10^{-4}. We also test the effect of varying the bond dimension.

4 Results

We trained the MPS-based model on three different datasets: MNIST, Fashion-MNIST, and CIFAR-10. Each of these datasets provides increasing complexity in terms of image features and pixel correlations. Our experiments were conducted by encoding each image into a quantum-inspired representation and processing it through the MPS classifier. The training and evaluation were conducted using a custom JAX implementation, which allowed us to leverage automatic differentiation and efficient tensor contractions on GPU.

On the MNIST dataset, which contains grayscale handwritten digits, the MPS classifier achieved strong results. With a bond dimension of 20, we obtained a final test accuracy of 98.9%, while training accuracy reached 100% within 15 epochs. These results suggest that the MPS model is capable of capturing the underlying structure of the MNIST images very effectively. We also observed that as the bond dimension increased from 2 to 20, the test accuracy steadily improved, but with diminishing returns beyond dimension 16. Increasing the bond dimension further, up to 40, only yielded marginal gains while significantly increasing memory usage and computation time per epoch.

On FashionMNIST, the performance remained competitive. With a bond dimension of 20, the model achieved a test accuracy of 90.5%. The convergence was slower compared to MNIST, requiring around 20 epochs for the model to plateau. Training accuracy reached approximately 97%, indicating some underfitting likely due to the limited expressivity of the MPS structure. These results were consistent across multiple runs, indicating the stability of training. Compared to traditional fully connected neural networks trained on the same dataset, the MPS model was slightly slower per epoch but showed better parameter efficiency for smaller bond dimensions.

For the CIFAR-10 dataset, which includes colored images with more complex textures and higher inter-pixel correlation, the MPS model struggled to achieve high accuracy. We processed the RGB channels by flattening the 32x32x3 image into a single vector of 1024 pixels and encoding each pixel into a qubit. With a bond dimension of 20, the model reached a final test accuracy of 46.2%. Training accuracy plateaued around 70%, showing that the model was able to learn some patterns but could not capture enough detail to compete with deep convolutional neural networks. The limitations in expressivity of the real-valued MPS and the absence of spatial inductive biases (such as those in CNNs) likely contributed to the reduced performance.

We also tested the impact of increasing the bond dimension on CIFAR-10, and while accuracy increased slightly up to a bond dimension of 40 (reaching 48%), the training time per epoch increased significantly, and memory usage became a bottleneck on our available hardware. This experiment confirms that

MPS-based classifiers are more effective for simpler, lower-dimensional data distributions and currently lack the structural capacity to handle complex visual datasets like CIFAR-10 without additional architectural innovations.

5 Conclusion & Outlook

In this paper, we explored the potential of Matrix Product State (MPS) models for quantum machine learning, specifically applied to classical image classification tasks. Our goal was to evaluate the feasibility of simulating quantum-inspired learning algorithms on classical hardware using tensor networks and to understand their performance across datasets of increasing complexity. We implemented a fully functional MPS classifier using JAX, which allowed us to efficiently perform automatic differentiation and optimization in a hardware-accelerated environment.

Our results demonstrate that MPS classifiers are highly capable on simple datasets like MNIST, where they can achieve near state-of-the-art accuracy with relatively small bond dimensions. On FashionMNIST, the model continues to perform well, with results comparable to shallow neural networks. However, on more complex datasets like CIFAR-10, the MPS model's performance drops significantly. This suggests that while MPS-based models are expressive enough for linearly structured or low-dimensional data, they may lack the representational capacity needed to capture hierarchical features present in natural images.

Despite these limitations, MPS remains a promising approach for quantum machine learning in the NISQ era. The architecture's compatibility with quantum circuits means that models trained classically could potentially be executed on actual quantum devices with shallow depth and limited resources. Moreover, the structure of MPS enables systematic scalability by increasing the bond dimension, providing a tunable tradeoff between computational cost and accuracy.

Future directions include extending the model to support complex-valued tensors, which would allow for simulation of full quantum states, capturing both magnitude and phase information. This would better align the classical model with true quantum behavior and may improve learning capacity. Additionally, introducing data-specific inductive biases — such as weight sharing or convolutional structures — into the MPS framework could help the model generalize better to high-dimensional data.

Another significant avenue is deploying the trained MPS on real quantum hardware. Since MPS can be mapped to quantum circuits with logarithmic depth in the number of qubits, deploying them on NISQ devices may be practical. Such implementations would offer insights into how tensor network-based models perform under real quantum noise, and whether their classical performance translates into quantum advantage.

Ultimately, our work contributes to a growing body of research seeking viable machine learning applications on near-term quantum devices, helping bridge the gap between theoretical quantum advantage and practical deployment.

A Implementation Details

The implementation was done entirely in Python using the JAX library. JAX's functional programming style and just-in-time compilation (via XLA) allowed for efficient training on both CPUs and GPUs. We made use of 'jax.numpy' for all tensor operations and 'jax.grad' for computing gradients of the loss function with respect to the MPS tensor parameters.

All tensors in the MPS were initialized to be identity matrices with a small amount of Gaussian noise (1×10^{-4} standard deviation) to break symmetry and ensure effective learning. The classifier was trained using the Adam optimizer with a fixed learning rate of 1×10^{-4}, a batch size of 64, and training was conducted for 20 epochs for MNIST and FashionMNIST, and 30 epochs for CIFAR-10. Early stopping was applied based on validation accuracy to prevent overfitting.

To encode the input data into quantum states, each pixel value x (normalized between 0 and 1) was transformed via the mapping

$$|\psi\rangle = \cos\left(\frac{\pi}{2}x\right)|0\rangle + \sin\left(\frac{\pi}{2}x\right)|1\rangle,$$

which corresponds to rotating the qubit into a superposition based on pixel intensity. This mapping produced a two-dimensional vector per pixel, forming an input matrix of shape 2×784 for MNIST and FashionMNIST, and 2×1024 for CIFAR-10.

The MPS forward pass involved contracting the input vectors with the visible indices of the MPS tensors and then sequentially contracting along bond dimensions to collapse the network into a final vector of shape $(10,)$ — one entry for each class label. The log-softmax function was applied to produce class probabilities, and the negative log-likelihood loss was computed via cross-entropy.

We experimented with bond dimensions of 2, 4, 8, 16, 20, and 40, observing both accuracy and computation time across each setting. GPU acceleration was provided by NVIDIA CUDA-enabled devices, and we monitored memory usage closely, as higher bond dimensions led to rapid increases in RAM and VRAM consumption.

All experiments were run multiple times to ensure consistency, and the results reported in the main paper are averaged across three independent runs.

B Cross Entropy

The cross-entropy loss function used to train the MPS classifier is defined as follows. Let y be the true label, encoded as a one-hot vector, and let p be the predicted class probabilities after applying the log-softmax to the output tensor of the MPS:

$$\text{Loss}(y, p) = -\sum_{i=1}^{C} y_i \log(p_i),$$

where C is the number of classes (10 for MNIST, FashionMNIST, and CIFAR-10), and p_i is the predicted probability of class i. This loss encourages the model to assign high probability to the correct label and penalizes uncertain or incorrect predictions.